A Compiler for CPPNs: Transforming Phenotypic Descriptions Into Genotypic Representations

Sebastian Risi
Center for Computer Games Research
IT University of Copenhagen
Copenhagen, Denmark 2300
sebr@itu.dk

Abstract

Biologically-inspired AI methods like evolutionary algorithms have shown great promise in creating complex structures yet these structures still pale in comparison to their natural counterparts. The recently introduced generative encoding *compositional pattern producing networks* (CPPNs), which is based on the principles of how natural organisms develop, narrowed this gap by showing that it is possible to artificially evolve life-like patterns with regularities at a high-level of abstraction. As these generative and developmental systems (GDS) are asked to evolve increasingly complex structures, the question of how to start evolution from a promising part of the search space becomes more and more important. To address this challenge, we introduce the concept of a *CPPN-Compiler*, which allows the user to directly compile a high-level description of the desired starting structure *into the CPPN itself*. In this paper, as proof of concept, the CPPN-Compiler is able to generate CPPN-encoded representations from vector-based images that can serve as the starting point for further evolution. Importantly, the offspring of these compiled CPPNs show meaningful variations because they *directly embody* important domain-specific regularities like symmetry or repetition. Thus the results presented in this paper open up a new research direction in GDS, in which specialized CPPN-Compilers for different domains could help to overcome the black box of evolutionary optimization.

Introduction

An interesting question in evolutionary algorithms (EAs) is what prevents us from evolving artifacts that rival the functional and structural complexity seen in natural organisms. One well-known problem facing EAs is the problem of *deception*, which means that a mutation increases fitness but actually leads further away from the final objective. Additionally, the objective functions in evolutionary computation do not necessarily reward the intermediate stepping stones that lead to the objective and the more ambitious the objective, the harder it is to identify a priori these stepping stones (Lehman and Stanley 2011; Woolley and Stanley 2011).

While recent methods like novelty search (Lehman and Stanley 2011) have shown great promise in avoiding deception in a variety of different domains, they alone might fail if the tasks become prohibitively complex or the space of possible behaviors is to vast (Lehman and Stanley 2010). Therefore this paper explores a potentially complementary approach towards reaching more ambitious objectives, in which the experimenter can impart important domain knowledge directly into the initial genotypic representation. Seeding evolution with a bias towards certain types of structures means escaping the black box of evolutionary optimization to provide a kind of general domain knowledge.

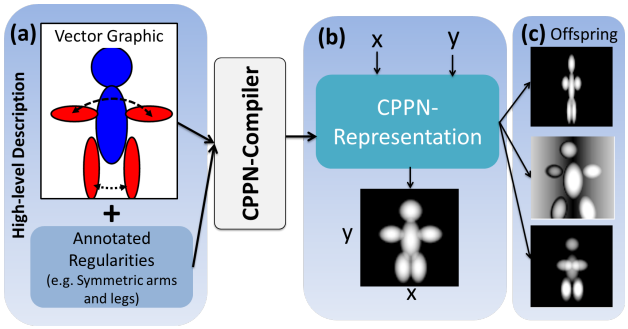


Figure 1: **CPPN-Compiler.** The compiler takes a high-level description as input (e.g. a vector image annotated with important regularities like symmetry) and compiles it into a CPPN (a). After compilation, the spatial patterns produced by the resulting CPPN show a clear resemblance to the original graphic (b). Because the compiled CPPN now directly embodies the annotated domain regularities (e.g. bilateral symmetric arms), the produced offspring show meaningful variations that nevertheless share common features (c).

While previous studies showed the promise of this approach, they require the manual construction of an appropriate evolutionary seed (Verbancsics and Stanley 2011; D'Ambrosio et al. 2010; Risi and Stanley 2012) or an external scaffold that evolution can elaborate on (Hoover, Szerlip, and Stanley 2011; Clune, Chen, and Lipson 2013).

In contrast, the approach in this paper takes first steps towards automatically creating an appropriate seed solely from a high-level domain-dependent description. Once such a seed is compiled, it does not depend on external inputs and directly embodies important domain-specific regularities. The promise of the new approach is that it could ultimately allow the creation of complex structures that start to resemble those in nature.

The possibility of evolving life-like patterns with regularities at a high-level of abstraction is created by the indirect encoding called *compositional pattern producing networks* (CPPNs; (Stanley 2007)). CPPNs have shown promise in a variety of different domains including the evolution of two-dimensional images (Secretan et al. 2011), three-dimensional forms (Clune and Lipson 2011), musical compositions (Hoover, Szerlip, and Stanley 2011; Hoover et al. 2012), robot morphologies (Risi, Cellucci, and Lipson 2013; Cheney et al. 2013; Auerbach and Bongard 2011), and the connectivity patterns of artificial brains (Stanley, D'Ambrosio, and Gauci 2009; Clune et al. 2009; Gauci and Stanley 2008; 2010). The idea behind CPPNs, which are an abstraction of natural development, is that patterns in nature can be described at a high level as compositions of functions, wherein each function in the composition represents a stage in development. Thus the indirect CPPN encoding can compactly encode patterns with regularities such as symmetry, repetition, and repetition with variation (Secretan et al. 2011; Stanley 2007).

In this paper, the introduced *CPPN-Compiler* allows the construction of the internal CPPN representation based on a high-level phenotypic description of the desired starting structure, instead of starting from random CPPN genotypes. While the system will be extended to more complicated domains in the future, the CPPN-Compiler concept is first explored through the interactive evolution of two-dimensional images. In this context, allowing evolution to start from a particular image (e.g. a butterfly, a car, etc.) and then further evolving it is an important research goal (Clune, Chen, and Lipson 2013).

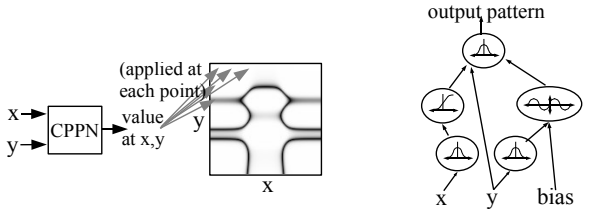
Figure 1 shows an example of how a vector-based image is converted into a CPPN-representation that is then evolved further. Different annotations allow the compiler to create a representation of the target image that *captures the underlying domain regularities* (e.g. bilateral symmetry), and *reuses* information to produce the final phenotype (e.g. both arms are encoded by the same genes).

The initial proof-of-concept results presented in this paper show that the patterns encoded by the compiled CPPNs do in fact resemble the patterns of the input images. Also importantly, because the internal CPPN representations are generated taking the domain regularities into account, the compiled representations display meaningful mutations (e.g. symmetric body parts show a coordinated change in size) and the ability to gracefully elaborate on the same theme.

Ultimately, the CPPN-Compiler is not restricted to creating images but can be extended to the wide variety of CPPN-enabled domains (Secretan et al. 2011; Clune and Lipson 2011; Hoover, Szerlip, and Stanley 2011; Hoover et al. 2012; Risi, Cellucci, and Lipson 2013; Cheney et al. 2013; Auerbach and Bongard 2011; Stanley, D'Ambrosio, and Gauci 2009), thus in effect opening up a new research direction in generative and developmental systems (GDS).

Background

This section provides an overview of CPPNs, which are capable of generating complex spatial patterns in Cartesian space, and then describes the NEAT method that is used to evolve them. Additionally, a review of prior work on seeding CPPNs manually is presented.



(a) Mapping (b) Composition

Figure 2: **CPPN Encoding.** (a) The function f takes arguments x and y, which are coordinates in a two-dimensional space. When all the coordinates are drawn with an intensity corresponding to the output of f, the result is a spatial pattern, which can be viewed as a phenotype whose genotype is f. (b) The CPPN is a graph that determines which functions are connected. The connections are weighted such that the output of a function is multiplied by the weight of its outgoing connection.

Compositional Pattern Producing Networks

The idea behind CPPNs is that patterns in nature can be described at a high level as compositions of functions, wherein each function in the composition represents a stage in development. CPPNs are similar to artificial neural networks (ANNs), but they rely on more than one activation function (each representing a common regularity).

The indirect CPPN encoding can compactly encode patterns with regularities such as symmetry, repetition, and repetition with variation (Secretan et al. 2011; Stanley 2007). For example, simply by including a Gaussian function, which is symmetric, the output pattern can become symmetric. A periodic function such as sine creates segmentation through repetition. Most importantly, *repetition with variation* (e.g. such as the fingers of the human hand) is easily discovered by combining regular coordinate frames (e.g. sine and Gaussian) with irregular ones (e.g. the asymmetric x-axis). For example, a function that takes as input the sum of a symmetric function and an asymmetric function outputs a pattern with imperfect symmetry. In this way, CPPNs produce regular patterns with subtle variations. The potential for CPPNs to represent patterns with motifs reminiscent of patterns in natural organisms has been demonstrated in several studies (Secretan et al. 2011; Stanley 2007).

Specifically, CPPNs produce a phenotype that is a function of n dimensions, where n is the number of dimensions in physical space. For each coordinate in that space, its level of expression is an output of the function that encodes the phenotype. Figure 2 shows how a two-dimensional phenotype can be generated by a function of two parameters that is represented by a network of composed functions.

Because CPPNs are a superset of traditional ANNs, which can approximate any function (Cybenko 1989), CPPNs are

also universal function approximators. Thus a CPPN can encode any pattern within its n-dimensional space.

In this paper, the initial CPPNs (which encode images) are *compiled* from a high-level phenotypic description, and then further evolved by NEAT, which is explained next.

Neuroevolution of Augmenting Topologies (NEAT)

NEAT typically starts with a population of simple neural networks and then *adds complexity* over generations by adding new nodes and connections through mutations. By evolving networks in this way, the topology of the network does not need to be known a priori; NEAT searches through increasingly complex networks to find a suitable level of complexity. For a complete overview of NEAT see Stanley and Miikkulainen (2002).

NEAT is chosen in this paper because it is proven effective for evolving ANNs and CPPNs (CPPN-NEAT) in a diversity of domains (Secretan et al. 2011; Hoover et al. 2012; Stanley and Miikkulainen 2002; Risi et al. 2012).

Manual CPPN Seeding and Functional Scaffolding

Because CPPN-NEAT evolves patterns as functions of geometry, general domain knowledge can be inserted into the evolutionary search algorithm. Seeding the initial CPPNs with a bias towards certain types of structures has shown promise in a variety of different domains, which are reviewed next.

One of the first examples of starting evolution with a manually constructed seed CPPN can be found in D'Ambrosio et al. (2010), in which the authors evolved multiagent teams encoded by CPPNs. In this setup CPPNs included some additional structure to start evolution with knowledge of mirrored team symmetry (i.e. the CPPN knows which side of the team is which and can opt to reflect one side across the midline), which helped to increase the overall performance of the teams.

In another promising demonstration, Verbancsics and Stanley (2011) showed that important general topographic principles for organizing connectivity can be seeded into the initial evolutionary population. Seeding CPPNs with a bias towards local connectivity, in which locality is expressed through a Gaussian function, allowed modular structures to arise naturally.

Risi and Stanley (2012) then demonstrated that it is not only possible to seed CPPNs with a bias towards local connectivity, but also to start the evolutionary search with a bias towards certain neural topographies (e.g. a locally-connected two-hidden-layer substrate). This advance was enabled by a method called ES-HyperNEAT (Risi and Stanley 2012), in which the CPPN can encode the connectivity, placement and density of the neurons in an artificial evolving neural network.

While these approaches highlight the intriguing potential of biasing the search towards certain types of structures, they required the manual and time-consuming construction of an appropriate seed. The approach presented in this paper tries to automate this process by directly constructing an appropriate start seed from a high-level description.

Instead of seeding the CPPN with a particular hidden node structure, it is also possible to instead input a *scaffold* into the CPPN, that evolution can then elaborate on. This related approach was first introduced by Hoover, Szerlip, and Stanley (2011), who showed that CPPNs can produce musical accompaniments to existing songs by exploiting regularities in the musical scaffold. Recently Clune, Chen, and Lipson (2013) adopted a similar approach that allows the further evolution of a given three-dimensional shape, by extending the CPPN with an additional input that describes the distance to the surface of the object. The image scaffold in Clune's work enabled morphing the input objects in interesting ways even though the initial CPPNs are flat (i.e. they do not contain any hidden nodes) and do not directly embody regularities in their *genotypes*. In other words, the *phenotypic* regularities are initially a result of the regularities in the input scaffold and not of the internal CPPN structure.

In the future, it will be interesting to compare or even combine the scaffolding approach with the method introduced in this paper; they both try to solve similar problems, but in significantly different ways.

Approach: CPPN-Compiler

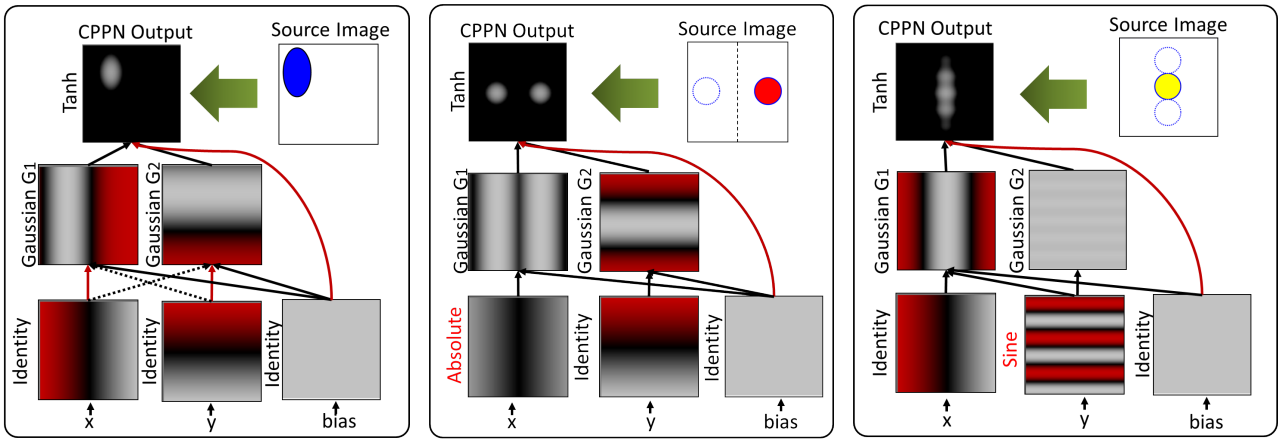
CPPN-NEAT typically starts with a randomly initialized population, in which the CPPNs have no hidden nodes and their inputs are directly connected with randomized weights to the outputs of the network. The NEAT algorithm then adds complexity over generations by adding new nodes and connections through mutations.

Instead of starting with random CPPNs, the presented approach allows the user to compile a high-level description of the desired starting structure into the CPPN itself (Figure 1). This advance is enabled by the CPPN's ability to efficiently encode regularities and should allow evolution to start from a promising part of the search space.

In this paper, the CPPN-Compiler concept is first explored through the interactive evolution of two-dimensional images. As highlighted by Stanley (2007), interactive evolution allows an easier first exploration of implicit encoding capabilities (e.g. elaboration and preservation of regularity) than a target-based evolution approach. In this context, allowing the user to start from a particular image and then further evolving it is an important research goal.

While Clune, Chen, and Lipson (2013) showed that phenotypic regularity can be input into the CPPN via an external scaffold, currently there exists no automated way of compiling images *together with their regularities* (e.g. symmetric wings on a butterfly) into the CPPN itself. The promise of such an approach is that the compiled CPPNs should share important properties with CPPN representations evolved from scratch, such as complexification in a way similar to natural organisms (Stanley 2007).

The basic idea of the CPPN-Compiler is that complex patterns with regularities can be represented by a few basic building blocks (Figure 1). Ellipses were chosen as the basic building block for the image to CPPN-Compiler in this paper because they lend themselves naturally to a CPPN representation and can be combined to resemble a variety of different



(a) Elliptical Shape (b) Bilateral Symmetry (c) Repetition

Figure 3: **Compiler Primitives.** The basic compiler building block is the ellipse, which can be represented by two overlapping Gaussian nodes (a). The compiler currently supports bilateral symmetry (shown in red), through the inclusion of an absolute function (b), and repetition (shown in yellow) encoded by a sine function (c). Positive CPPN output values are shown in white, while negative values are shown in red.

patterns. Each ellipse in the original graphic is compiled into the internal CPPN representation based on one of three possible annotations: *single entity*, *bilateral symmetric*, or part of a *repeating pattern*. The idea behind these annotations is that they allow the compiler to create a representation of the target image that *captures the underlying domain regularities* and *reuses* information to produce the final phenotype.

For example, in Figure 1 the two arms of the simplified body plan are annotated as being symmetric, which means the algorithm should *not* encode them separately in the CPPN; instead the compiler will build a CPPN that reuses information through inclusion of a symmetric function (explained below). Importantly, these annotations also allow coordinated and meaningful mutations on the generated structures (e.g. one mutation can increase the length of *both* arms simultaneously), therefore enabling an effective and intuitive exploration of the search space. The details of the CPPN-Compiler algorithm[1] are explained below and also depicted in Figures 3 and 4.

Compiling Elliptical Shapes: Each ellipse in the source image is defined by its position (lx, ly) scaled into the range $[-1, 1]$ and rotation r between $0°$ and $180°$ degrees. The main motivation for choosing ellipses as the basic building block for representing images is the insight that they can easily be represented by the overlap of two Gaussian functions G_1 and G_2 (Figure 3a) that receive input from x and y, respectively. The weights w_1 and w_2 from the bias node to Gaussian nodes G_1 and G_2 are set based on the location (lx, ly) of the ellipse. They are determined as follows: $w_1 = -lx$ and $w_2 = -ly$. For example, the further left the ellipse is located, the more positive the weight w_1 should become, moving the center of the CPPN-produced Gaussian pattern G_1 further to the left as well.

[1]A Python implementation of the CPPN-Compiler will soon be made available at http://www.cs.ucf.edu/ risi/

The width and height of each ellipse is encoded through the strength of the connections from x and y to the Gaussian nodes G_1 and G_2, respectively. Ellipses with a specific rotation r can also be compiled into the CPPN. If $r \neq 0.0$, additional connections are added from x to G_2 and from y to G_1 (denoted as dotted lines in Figure 3a). The previously determined weight of the connection from x to G_1 is scaled by $\frac{1+r}{180.0}$ and the weight of the connection from y to G_1 scaled by $\frac{-r}{180.0}$. The connections to G_2 are scaled accordingly but in reverse (i.e. from x by $\frac{-r}{180.0}$ and from y by $\frac{1+r}{180.0}$).

Finally, the output of the two Gaussian nodes and a negative connection from the bias feed into a strictly positive function f that is 0.0 for $f(x) < 0.0$ and $tanh(x)$ otherwise. That way resulting negative values are filtered out, allowing the CPPN to produce a smooth positive pattern (shown in white) for each ellipse that contrasts with the black pattern surrounding it.

Bilateral Symmetry: An important regularity in natural systems from faces to animals is symmetry. A bilateral symmetric ellipse is encoded inside the CPPN by feeding the absolute value of x into the Gaussian node G_1 instead of x directly (Figure 3b). Therefore, the same information is reused to encode the annotated ellipse and its bilateral symmetric counterpart.

Repeating Pattern: Another essential regularity seen in biological systems is repetition. Repetition can naturally be compiled into a CPPN by first applying a periodic sine function to the CPPN inputs before feeding them to the Gaussian nodes. This function than in turn produces a repetition of parts without the need to duplicate the information for each part separately in the CPPN. The frequency of the repetition is determined by an additional *frequency* annotation that determines the strength of the connection between the sine and the Gaussian node. Figure 3c shows an example of repetition along the y-axis that produces a segmentation-like pattern.

Linking Multiple Patterns: The CPPN-Compiler also supports linking multiple independently encoded patterns into modules, as seen in Figure 4. The motivation behind linking patterns in this way is to facilitate coordinated mutations among phenotypic parts that are not encoded by the same genotypic information. For instance in humans, red haired people are far more likely to also have freckles then people with black hair because of a phenomena called *linkage disequilibrium* (Reich et al. 2001), which means there is a nonrandom association of the alleles encoding these traits.

Linkage is encoded in the CPPN genotype by grouping all the linked objects together in one module. Each of these modules replicates the x, y and $bias$ inputs, which are connected to the original CPPN inputs. By encapsulating objects in this way, a single mutation on the connections from the input nodes can affect all encapsulated patterns simultaneously. For example, changing the weight of the single connection C (highlighted in Figure 4) would scale the width of the two encapsulated ellipses.

The next section describes the interactive evolution experiments in this paper, which are designed to explore the capabilities of the presented CPPN-Compiler.

Experiment

The experiments in this paper are performed in a modified version of SharpNEAT-based Genetic Art (SNGA) platform by Holger Ferstl, which makes it possible to explore how evolution establishes and varies regularities. In this setup the experimenter picks the parent(s) of the next generation by clicking on them, which means that the selection process is entirely guided by the human experimenter. Each new generation is then created by mutating the parent generation through the NEAT algorithm (Stanley and Miikkulainen 2002; 2004).

The CPPN-Compiler is tested on three very simplified vector graphic representations of a face, a human body, and an insect (Figure 5,top). These images are chosen to test the CPPN-Compiler on increasingly difficult regularities. The face is designed to be easy to represent with one symmetric eye region (marked in red). The human body is slightly more complicated with two symmetric regions (legs, arms). Finally, the insect combines both bilateral symmetry (wings) and repetition along the y-axis (body segmentation shown in yellow). Together these images will test the ability of the CPPN-Compiler to (1) reproduce the input patterns by a CPPN, and (2) display meaningful mutations and the ability to gracefully elaborate on the compiled representations.

Experimental Parameters

Because CPPN-NEAT differs from original NEAT only in its set of activation functions, it uses the same parameters (Stanley and Miikkulainen 2002). Offspring had a 0.90 probability of link weight mutation, 0.45 chance of link addition, and 0.5 chance of node addition. The available CPPN activation functions were sigmoid, Gaussian, absolute value, cosine, and sine, all with equal probability of being added. Parameter settings are based on prior reported settings for NEAT (Stanley and Miikkulainen 2002; 2004; Stanley 2007). They were found to be robust to moderate variation through preliminary experimentation.

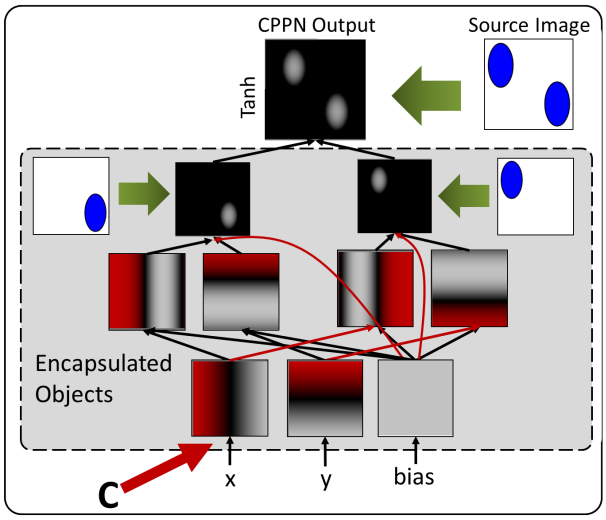


Figure 4: **Linking Multiple Patterns.** The CPPN-Compiler supports linking multiple independently encoded patterns into one module. Each of these modules replicates the x, y and $bias$ inputs, which are connected to the original CPPN inputs. By encapsulating objects in this way, a single mutation on the connections from the original input nodes can affect each object simultaneously (e.g. changing the weight of the single connection C can scale the width of the two encapsulated ellipses).

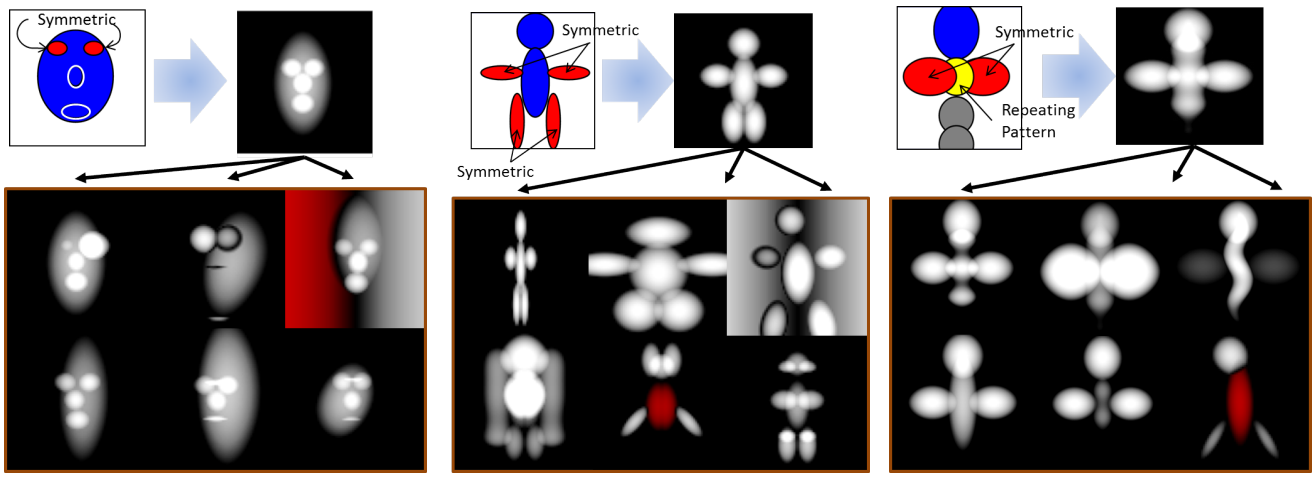
Results

Figure 5 shows the three test images (face, human body, insect) that are compiled into the CPPN representations and then further evolved. The patterns produced by the compiled CPPNs show a clear resemblance to the original vector graphics (Figure 5,top).

In general the regularities compiled into the CPPN allow the user to successfully elaborate on the same theme (Figure 5,bottom). For example, a variety of insect patterns with differently shaped wings and body segmentations, human bodies with different proportions, and variations on face-like patterns are evolved.

Because NEAT adds complexity by adding new nodes and connections through mutations, the images can become more complex over generations. Elaboration through complexification can be observed in the appearance of imperfect symmetry (the eyes in Figure 5a) or the appearance of glasses. Additionally, while some mutations produce minor changes in the resulting phenotypes (Figure 5b,top), other mutations produce a greater variety of different body plans (Figure 5b,bottom) that nevertheless share common features.

The first two descendants in Figure 5a show an example of linkage. Changing the weight of a single connection that connects the encapsulated objects to the x CPPN input determines the width of the body. In effect, it is like a gene that affects the body proportions. In this way, the compiled-CPPNs create meaningful mutations of the source images.



(a) Face (b) Human Body (c) Segmented Insect

Figure 5: **Compiled CPPN Patterns and Evolved Descendants.** The patterns produced by the compiled CPPNs show a clear resemblance to the input graphics (top). The offspring of the compiled CPPNs exhibit graceful mutations with variations and elaborations on the same theme.

The main result is that it now seems feasible to jump-start evolution from an interesting part of the CPPN-encoded pattern space. This advance is important because the common practice of starting with random genotypes has likely limited EAs in solving highly ambitious problems in the past.

Discussion and Future Work

The more ambitious the objective, the more likely it is that search can be deceived by suboptimal solutions (Lehman and Stanley 2011; Woolley and Stanley 2011). Thus it becomes increasingly important to be able to start evolution from a promising part of the search space, from which the final objective might be easier to reach. While previous studies showed the promise of seeding evolution with a bias towards certain types of structures (Verbancsics and Stanley 2011; D'Ambrosio et al. 2010; Risi and Stanley 2012), they required the manual construction of an appropriate CPPN seed, which is a time consuming and potentially error prone task. This study presented first results towards automating the construction of an appropriate seed that directly embodies important domain-specific regularities like symmetry and repetition.

While the domain in this paper is simple, it nevertheless showed that it is possible to start evolution from an interesting part of the CPPN-encoded pattern space through the automated compilation of annotated vector graphics. Importantly, the offspring of the compiled genomes still display the same key properties (e.g. elaborations on the same theme, complexification, symmetric and asymmetric alterations, etc.) found in CPPN representations evolved from scratch (Stanley 2007), thus corroborating the choice in compiler primitives.

Also interestingly, some regularities like bilateral symmetry seem to have a high level of preservation, which is also a well observed phenomena in natural organisms. Even though mutations on the compiled CPPNs sometimes produce offspring that lose symmetry, most offspring preserve the bilateral symmetry while still allowing the elaboration of the limb design itself. Analyzing the exact reason for this preservation and supporting quantitative results will be an important future research direction.

Thus now that we know it is possible to automate the process of seeding evolution with a bias towards certain structures, a further important task is to apply this approach to more complicated domains. For example, CPPNs also enable a method called Hypercube-based NeuroEvolution of Augmenting Topologies (HyperNEAT; (Stanley, D'Ambrosio, and Gauci 2009; Gauci and Stanley 2008; 2010)) to evolve large-scale ANN controllers. Based on the insight that a recent HyperNEAT extension called ES-HyperNEAT (Risi and Stanley 2012) can be seeded with a bias towards certain neural topologies, the presented approach opens up the intriguing possibility of a *neural CPPN-Compiler*. Such a compiler could automatically construct CPPNs that produce structural organization we see in natural brains (e.g. the organization of the cortical columns), potentially allowing higher cognitive tasks to be solved.

Conclusion

This paper presented a novel approach that can automatically generate CPPN-encoded representations from high-level phenotypic descriptions. As proof of concept, the CPPN-Compiler is able to generate genotypic representations of vector-based images that are then evolved further. Importantly, the offspring of these compiled CPPNs show meaningful variations because they directly embody important domain-specific regularities like symmetry and repetition. In the future this approach could be extended to a variety of other domains; especially a neural CPPN-Compiler that could allow higher cognitive functions to evolve is now

an intriguing possibility. The harder the domain the more important it will be to enable the user to automatically jumpstart evolution from a promising starting point and to bias the search towards certain types of structures.

Acknowledgments

This work was supported by the Postdoc-Program of the German Academic Exchange Service (DAAD).

References

Auerbach, J. E., and Bongard, J. C. 2011. Evolving complete robots with cppn-neat: the utility of recurrent connections. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, 1475–1482. ACM.

Cheney, N.; MacCurdy, R.; Clune, J.; and Lipson, H. 2013. Unshackling evolution: Evolving soft robots with multiple materials and a powerful generative encoding. In *Proceedings of the Genetic and Evolutionary Computation Conference*.

Clune, J., and Lipson, H. 2011. Evolving 3d objects with a generative encoding inspired by developmental biology. In *Proceedings of the European Conference on Artificial Life (Alife-2011)*, volume 5, 2–12. New York, NY, USA: ACM.

Clune, J.; Beckmann, B. E.; Ofria, C.; and Pennock, R. T. 2009. Evolving coordinated quadruped gaits with the HyperNEAT generative encoding. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC-2009) Special Section on Evolutionary Robotics*. NJ, USA: IEEE Press.

Clune, J.; Chen, A.; and Lipson, H. 2013. Upload any object and evolve it: Injecting complex geometric patterns into cppns for further evolution. In *Proceedings of the IEEE Congress on Evolutionary Computation*.

Cybenko, G. 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems* 2(4):303–314.

D'Ambrosio, D. B.; Lehman, J.; Risi, S.; and Stanley, K. O. 2010. Evolving policy geometry for scalable multiagent learning. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, 731–738. International Foundation for Autonomous Agents and Multiagent Systems.

Gauci, J., and Stanley, K. O. 2008. A case study on the critical role of geometric regularity in machine learning. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-2008)*. Menlo Park, CA: AAAI Press.

Gauci, J., and Stanley, K. O. 2010. Autonomous evolution of topographic regularities in artificial neural networks. *Neural Comput.* 22:1860–1898.

Hoover, A. K.; Szerlip, P. A.; Norton, M. E.; Brindle, T. A.; Merritt, Z.; and Stanley, K. O. 2012. Generating a complete multipart musical composition from a single monophonic melody with functional scaffolding. In *Proceedings of the International Conference on Computational Creativity (ICCC-2012). Dublin, Ireland*.

Hoover, A. K.; Szerlip, P. A.; and Stanley, K. O. 2011. Interactively evolving harmonies through functional scaffolding. In *Proceedings of the Genectic and Evolutionary Computation Conference (GECCO-2011). New York, NY: The Association for Computing Machinery*.

Lehman, J., and Stanley, K. O. 2010. Revising the evolutionary computation abstraction: Minimal criteria novelty search. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2010)*. ACM.

Lehman, J., and Stanley, K. O. 2011. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation* 19(2):189–223.

Reich, D. E.; Cargill, M.; Bolk, S.; Ireland, J.; Sabeti, P. C.; Richter, D. J.; Lavery, T.; Kouyoumjian, R.; Farhadian, S. F.; Ward, R.; et al. 2001. Linkage disequilibrium in the human genome. *Nature* 411(6834):199–204.

Risi, S., and Stanley, K. O. 2012. An enhanced hypercube-based encoding for evolving the placement, density, and connectivity of neurons. *Artificial Life* 18(4):331–363.

Risi, S.; Lehman, J.; D'Ambrosio, D. B.; Hall, R.; and Stanley, K. O. 2012. Combining search-based procedural content generation and social gaming in the petalz video game. In *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2012)*.

Risi, S.; Cellucci, D.; and Lipson, H. 2013. Ribosomal robots: Evolved designs inspired by protein folding. In *Proceedings of the Genetic and Evolutionary Computation Conference*.

Secretan, J.; Beato, N.; D'Ambrosio, D.; Rodriguez, A.; Campbell, A.; Folsom-Kovarik, J.; and Stanley, K. 2011. Picbreeder: A case study in collaborative evolutionary exploration of design space. *Evolutionary Computation* 19(3):373–403.

Stanley, K. O., and Miikkulainen, R. 2002. Evolving neural networks through augmenting topologies. *Evolutionary Computation* 10:99–127.

Stanley, K. O., and Miikkulainen, R. 2004. Competitive coevolution through evolutionary complexification. *JAIR* 21:63–100.

Stanley, K. O.; D'Ambrosio, D. B.; and Gauci, J. 2009. A hypercube-based indirect encoding for evolving large-scale neural networks. *Artificial Life* 15(2):185–212.

Stanley, K. O. 2007. Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines Special Issue on Developmental Systems* 8(2):131–162.

Verbancsics, P., and Stanley, K. O. 2011. Constraining Connectivity to Encourage Modularity in HyperNEAT. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2011)*. New York, NY: ACM.

Woolley, B. G., and Stanley, K. O. 2011. On the deleterious effects of a priori objectives on evolution and representation. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, 957–964. ACM.